MASTER EN SCIENCES INFORMATIQUES

Conception et réalisation d'adaptateurs d'interfaces intelligents

Claes, Luc

Award date:
1982

Awarding institution:
Universite de Namur

Facultés Universitaires Notre-Dame de la Paix, Namur
Institut d'Informatique
Année Académique 1981-1982

Conception et Réalisation d'adaptateurs d'interfaces intelligents

Luc CLAES

Mémoire présenté en vue de l'obtention du grade de licencié et Maître en informatique.

TABLE DES MATIERES

AVANT PROPOS

Je tiens à remercier Messieurs Philippe VAN BASTELAER, Claude CHERTON et Joseph DEMARTEAU pour l'intérêt qu'ils ont porté à la réalisation de ce travail.

Je remercie également Mademoiselle Anne BAIJOT, la "fée" du traitement de texte.

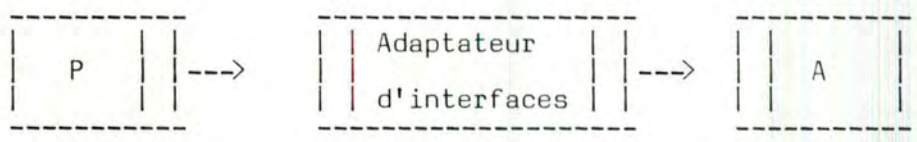
INTRODUCTION

Introduction

Les pages qui suivent font largement appel à un substantif totalement absent de tous nos grands dictionnaires, mais dont l'usage est largement répandu dans le monde de l'informatique, j'ai cité l'interface.

Voici une définition imagée de ce concept :
soit deux systèmes de traitement de données qui seront appelés A (l'auditeur) et P (le parleur). P peut émettre des données vers l'extérieur, A peut "écouter" des données venant de l'extérieur. Ces deux systèmes possèdent donc une jonction avec le monde extérieur leur permettant de communiquer avec lui. L'ensemble des caractéristiques de cette jonction sera appelé interface de communication.

Le transfert de données de P à A n'est possible que si leurs interfaces sont compatibles. Si ce n'est pas le cas, il sera nécessaire de mettre en oeuvre un troisième système disposant de deux interfaces, l'un compatible à P et l'autre compatible à A. Ce système sera appelé adaptateur d'interfaces.



Ainsi, un modem, un processeur frontal sont des adaptateurs d'interfaces.

Nous parlerons ici uniquement des adaptateurs d'interface "intelligents", c'est-à-dire ne se limitant pas à des adaptations purement physiques.

Que cela se passe en haute fidélité, en plomberie, en électricité ou, bien sûr, en informatique, partout où la technique humaine est à l'oeuvre, des problèmes d'adaptation, d'incommunicabilité existent. Ils sont le fruit amer de la créativité, mais aussi d'une certaine volonté d'indépendance ou, plus prosaïquement, d'une motivation purement commerciale.

Ce n'est pas l'anarchie non plus. La volonté de normalisation existe partout, à en juger par l'omniprésence des DIN, ISO, CCITT, EIA et autres IEEE.

Nous naviguons quelque part, entre l'ennui de l'uniformité et le danger du chaos; l'équilibre sans doute ? Le non-respect de normes est souvent compréhensible mais nécessite toujours la mise en oeuvre d'une fonction d'adaptation plus ou moins complexe.

Historiquement, notre première prise de contact avec le problème des adaptateurs d'interfaces date de fin 1978. Il s'agissait alors de supprimer les fastidieuses opérations d'encodage manuel de résultats provenant d'un auto-analyseur médical, en réalisant une connexion directe entre cet auto-analyseur et un ordinateur central. La connexion directe s'avèra vite impossible compte tenu des problèmes posés parmi lesquels nous relèverons :

- l'incompatibilité des horaires de travail de l'ordinateur central et de l'auto-analyseur, ce dernier étant susceptible de travailler de nuit et ne disposant d'aucun organe de stockage,

- la nécessité, imposée par le logiciel de base de l'ordinateur, d'initialiser la saisie de données par une séquence de commandes (identification de l'utilisateur mot de passe, appel d'un programme spécifique),

- l'importance, transitoirement très élevée, du débit d'informations produites par l'auto-analyseur,

- la présence, au sein des messages de produits par l'auto-analyseur, de caractères spéciaux incompatibles quant à leur interprétation par l'ordinateur,

- la rigidité des logiciels de base des deux protagonistes.

Ces problèmes mirent en évidence la nécessité de mise en oeuvre d'un organe intermédiaire chargé d'une double fonction : le stockage de données et l'adaptation d'interfaces proprement dite.

Le nombre et la complexité des tâches à remplir par cet organe nous amena à concevoir une architecture matérielle basée sur un microprocesseur.

Les premiers temps furent rudes ! Nous avons réinventé la roue : assemblage manuel, introduction manuelle de (longs !) programmes en hexadécimal, découverte du microprocesseur et de ses coupleurs d'entrées/sorties ... L'aventure. Ce premier adaptateur d'interface fut réalisé en six mois.

Par la suite, nous avons été amenés à résoudre de nombreux autres problèmes d'interconnexion qui nous ont amené à définir une découpe fonctionnelle "standard" des adaptateurs d'interface et à créer un ensemble d'outils aidant à la réalisation de nouvelles applications.

La mise en oeuvre de ces outils permet de réaliser un adaptateur d'interface non plus en six mois, mais bien en quelques heures ou quelques jours. Ils ne représentent pas une panacée, mais ils offrent l'avantage de nous consacrer presque uniquement sur les aspects particuliers à chaque nouvelle application.

Signalons déjà ici qu'un des problèmes majeurs que l'on rencontre lors de la conception de logiciels d'interfaçage réside dans les nombreuses lacunes des spécifications concernant les interfaces à adapter.

Le réalisateur de la fonction d'adaptation est amené à découvrir ces spécifications par l'expérience, le constructeur de l'appareil à adapter ne fournissant très souvent que des informations incomplètes ou erronnées. Voilà de quoi faire bondir plus d'un informaticien !

Les annexes à ce mémoire, comprenant un manuel d'utilisateur ainsi que les programmes - sources d'un logiciel d'adaptateur d'interface ne sont accessibles qu'au secrétariat de l'Institut d'Informatique de Namur; ces annexes constituant, à l'heure actuelle, un produit commercialisé.

Les pages qui suivent ne reflètent pas une vue "a priori", une pré-étude du problème des adaptateurs d'interfaces, mais elles résultent seulement de l'expérience.

Ce mémoire est structuré en trois parties :

I **Aspects fonctionnels des adaptateurs d'interfaces**

II **L'architecture du logiciel**

III **Réalisation des adaptateurs d'interfaces**

I ASPECTS FONCTIONNELS DES ADAPTATEURS D'INTERFACES

I. Aspects Fonctionnels des Adaptateurs d'Interfaces

1.1. Généralités

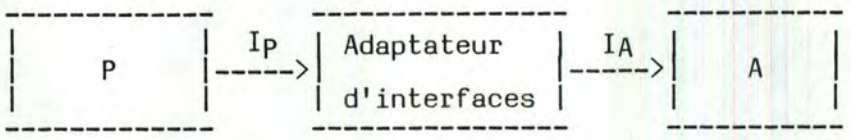
Afin de mettre en évidence les concepts fondamentaux liés à l'adaptation d'interfaces, nous utiliserons un modèle mettant en jeu deux systèmes : P (le parleur) et A (l'auditeur).

Le système P peut émettre un flux d'informations vers le monde extérieur. L'ensemble des caractéristiques de ce flux sera appelé l'interface de P, soit I_P.

Le système A peut recevoir un flux d'informations venant du monde extérieur. Il ne peut cependant accepter qu'un flux ayant un ensemble de caractéristiques donné appelé l'interface de A, soit I_A

L'adaptation d'interface sera nécessaire si les interfaces I_P et I_A sont incompatibles.

Nous définirons l'adaptateur d'interface come un système transformant un flux d'informations de caractéristiques I_P en un flux d'informations de caractéristiques I_A.



Nous parlerons d'adaptateur d'interface intelligent lorsque l'incompatibilité à résoudre entre I_P et I_A se situe ailleurs qu'au niveau physique et ne peut être abordée de manière purement combinatoire.

Dans un but simplificateur, nous nous limiterons aux communications uni-directionnelles, ne mettant en jeu qu'un seul parleur et un seul auditeur.

Les problèmes d'adaptation principaux sont les suivants:

- A n'écoute P qu'à certains moments de la journée, mais ne veut rien perdre de ce qu'a dit P.
- A et P utilisent des langages différents (mais utilisent les mêmes unités sémantiques).
- A et P ne peuvent pas utiliser les mêmes supports de communication.
- A n'est sûr des informations de P que si elles sont suivies de symboles de vérification.
- Le chemin de communication entre P et A peut engendrer du bruit.

En résumé, le problème consiste à rendre, non seulement possible mais aussi déterministe, la communication de P à A en tenant compte de leurs "personnalités" respectives.

Il n'est pas question ici d'analyser un couple (P, A) en particulier, mais bien d'établir une découpe fonctionnelle suffisamment générale pour s'adapter à presque tous les couples (P, A) existant à l'heure actuelle. En effet, lorsque l'on analyse un problème particulier, on découvre un ensemble de spécifications qui peut se décomposer comme suit :

- spécifications liées uniquement à l'auditeur : S(A)

- spécifications liées uniquement au parleur : S(P),

- spécifications liées à l'auditeur et au parleur : S(P, A).

Cette partition des spécifications a une portée intéressante au niveau organique, où elle offre un critère de décomposition en modules, tout en évitant nombre de dangereuses redondances.

Comme nous nous intéressons tout particulièrement aux adaptateurs d'interfaces intelligents, la réalisation de l'adaptation fera toujours appel à un micro-ordinateur.

L'architecture matérielle de ces ordinateurs est susceptible de nombreuses modifications d'une application à l'autre. Nous verrons donc l'apparition de spécifications S(H), S(A,H), S(P,H) et S(P,A,H), (H pour hardware).

Ainsi, la description générale du matériel sera reprise dans S(H) (configuration mémoire, description des coupleurs d'entrées/sorties) et les paramètres des coupleurs d'entrées/sorties propres à un auditeur particuliers seront repris dans S(A,H).

Nous effectuerons d'abord une démarche d'analyse fonctionnelle "générale", qui peut s'adapter à un grand nombre d'interfaces pour ensuite donner des éléments d'analyse fonctionnelle particulière à un interface donné.

1.2. L'être humain comme adaptateur d'interface

Lorsqu'une forte incompatibilité existe entre les moyens de communication de P et de A, l'adaptateur d'interface utilisé est encore souvent l'être humain. Doué d'une faculté d'adaptation extraordinaire, il est souvent employé comme intermédiaire entre une source d'informations et un système informatique. Quand la source d'informations est elle-même un appareillage de traitement de données, l'interface humain offre de nombreux inconvénients :

- lenteur du transfert de données,
- introduction importante et difficilement mesurable de bruit dans la communication (fautes de frappe par exemple),
- nécessité de lui offrir un dialogue de haut niveau "en clair",
- coût souvent élevé.

Ces inconvénients peuvent souvent être éliminés par l'emploi d'un adaptateur d'interface intelligent. Remarquons que le remplacement de l'être humain par l'adaptateur d'interface intelligent peut avoir d'importantes implications sociales. Ici comme ailleurs, les emplois visés sont ceux faisant appel à l'automatisme plutôt qu'à la créativité.

1.3. L'ordinateur central sans adaptateur d'interface

Entendez par ce titre de paragraphe :

"Un ordinateur central puissant a-t-il besoin d'aide extérieure pour acquérir des données ? Ne dispose-t-il pas presque toujours d'une gamme très large de contrôleurs faisant office d'adaptateur d'interface ?"

Pour répondre à cette question, il faut envisager cinq types de critères qui sont :

1o Les possibilités offertes par le logiciel de l'ordinateur et en particulier par son logiciel de base.

2o Les possibilités offertes par le matériel de l'ordinateur pour les communications de données avec l'extérieur.

3o L'activité globale du système : charge de travail, fonctionnement permanent ou non, risques de pannes .

4o La difficulté de mise en oeuvre de l'adaptation.

5o Les coûts entraînés par la réalisation de l'adaptation.

En général, lorsque l'on s'éloigne d'interfaces standards, l'ordinateur central sans adaptateur d'interface s'avère être une solution coûteuse, quand elle n'est pas techniquement irréalisable.

1.4. Etude des problèmes d'adaptation

1.4.1. Incompatibilités Auditeur/Parleur

1. Les systèmes d'exploitation des ordinateurs travaillent aujourd'hui presque exclusivement en temps partagé, plus rarement en traitement par lots, mais presque jamais en "temps réel" au sens strict de ce terme. La majeure partie des appareils automatiques susceptibles de communiquer avec lui envoient leurs messages "à l'aveugle", sans se soucier d'une acceptation de la part de leur interlocuteur. Si l'ordinateur n'est pas en permanence à l'écoute des messages ou n'est pas capable de traiter ces messages en temps voulu, une bonne partie d'entre eux ne seront jamais pris en compte. Nous avons donc affaire à une communication aléatoire (parfois le message sera bien reçu, parfois non).

   L'adaptateur d'interface aura donc dans ses attributions de rendre la communication déterministe, quitte à établir un protocole avec l'ordinateur, avec accusé de réception, demande de répétition, etc ...

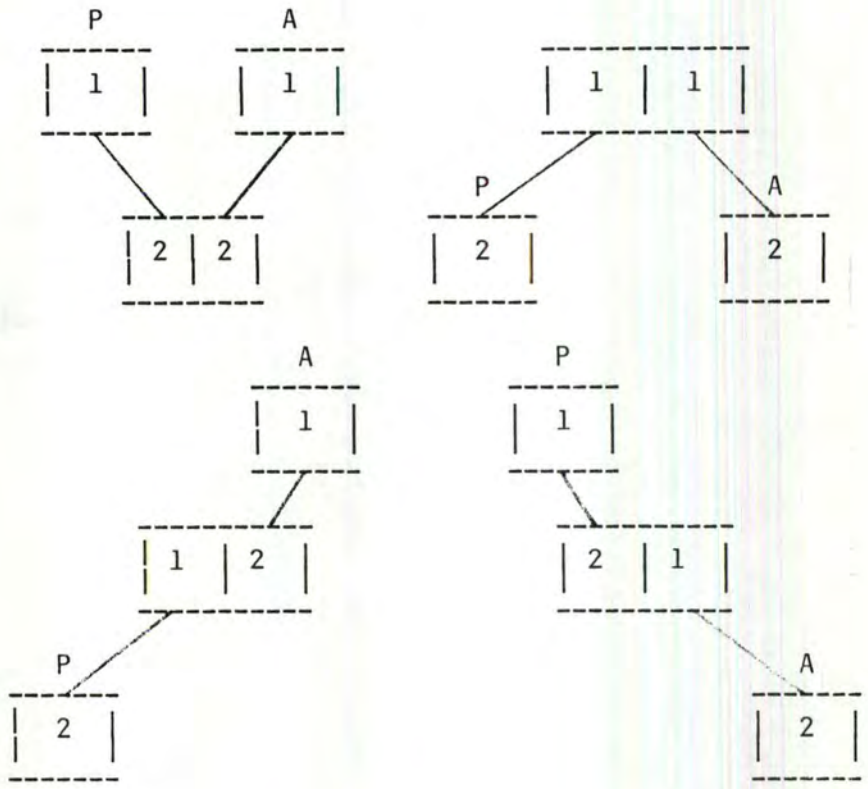
2. Le débit d'information produit par P peut être supérieur, transitoirement, à la capacité de réception de A. L'adaptateur devra donc mémoriser les informations pour faire office de tampon entre P et A. Il pourra également filtrer le flux d'information venant de P pour en ôter certaines redondances.

3. Les logiciels de base ne permettent que rarement la transparence totale des symboles sur un canal d'entrée. Certains symboles y sont interprétés comme, par exemple, caractères de blocage/déblocage de ligne (X-ON, X-OFF), caractères de fin de processus, etc...
Si ces symboles sont générés innocemment, et dans un tout autre but, par P, des catastrophes sont à craindre. L'interface devra donc filtrer ou transcoder ces caractères.

4. Le logiciel de base de A peut n'avoir comme notion de canal d'acquisition de données que le "terminal", clavier/écran ou clavier/imprimante. Ne peuvent dès lors accéder au système que les "êtres" capables d'exécuter des commandes de login/mot de passe, identification d'utilisateur, ... pour ensuite demander l'exécution de programmes. L'adaptateur d'interface devra donc parfois se faire passer pour un terminal manipulé par un utilisateur humain.

5. La communication uni-directionnelle de P vers A implique qu'un des deux systèmes soit une station primaire et l'autre une station secondaire. Rappelons qu'une station primaire est susceptible d'émettre des messages ou des commandes spontanément, alors qu'une station secondaire ne fait que répondre à des commandes. Ainsi, un terminal asynchrone sera souvent une station primaire alors qu'un terminal synchrone "pollé" sera une station secondaire.

   De nombreux logiciels de base d'auditeurs ou de parleurs ne disposent que d'un seul des deux modes de fonctionnement. L'adaptateur d'interface devra donc également adapter les stations à ce niveau. Les quatre possibilités à envisager sont représentées ci-après, où "1" représente une fonction primaire et "2" une fonction secondaire.



Le lecteur trouvera une description générale des fonctions primaires et secondaires dans **<2>**.

Notons que ces concepts à eux seuls peuvent présenter une grande complexité, surtout lorsque l'on retire l'hypothèse d'une communication uni-directionnelle.

6. Il est souvent souhaitable, lorsque plusieurs adaptateurs d'interfaces sont connectés sur un même système de banaliser les formats de messages dans une large mesure. Les programmes d'application peuvent ainsi faire appel à un unique programme de réception, indépendant des différents parleurs.

7. P ne fournit que rarement une protection contre le bruit introduit dans la communication. l'interface situé physiquement à proximité de P peut générer des mécanismes de détection d'erreurs et rendre la communication plus fiable.

1.4.2. Activité globale du système auditeur

A l'heure actuelle, nombre d'ordinateurs sont appelés à effectuer des tâches multiples quasi simultanément. La centralisation a encore de beaux jours devant elle...
Elle pose également de gros problèmes dans le cas qui nous occupe :

1. Les temps de réponse peuvent varier très sensiblement. Il n'est pas question de travailler "à l'aveugle"; il faut tenir compte de la notion de temps partagé par de nombreuses tâches. (voir 1.4.1.1.)

2. Les pannes d'un système informatique centralisé sont souvent de petites catastrophes ... l'adaptateur d'interface doit veiller à en atténuer les effets (par exemple en évitant la réintroduction manuelle de données.)

3. Le fonctionnement d'un système central de moyenne importance est rarement permanent. Mais il n'est pas rare de voir des appareils automatiques débiter des informations pendant la nuit.

Ces quelques points mettent en évidence une fonction primordiale, déjà citée, de mémorisation à effectuer par l'adaptateur d'interface. Plus qu'un simple tampon régulateur de débit, l'adaptateur devra souvent être capable d'emmagasiner un grand volume de données.

1.4.3. Limitations matérielles de l'auditeur

Sans entrer dans des considérations tenant plus à l'électronique qu'à l'informatique, il faut cependant tenir compte des nombreuses manières de représenter physiquement un "bit", puis un symbole.

Le bit est toujours défini par une variation de grandeur physique dans le temps; les choix offerts, en ce domaine, par la technologie actuelle sont très nombreux.

Souvent, les bits doivent encore être structurés en caractères. Les règles de structuration existantes sont, elles aussi, très nombreuses.

L'auditeur devrait donc pouvoir faire face à des choix de paramètres comme par exemple :

- vitesse de transmission,
- transmission parallèle/série,
- mode synchrone, asynchrone,
- modulation éventuelle AM/FM/PM,
- transmission optique/électrique/électromagnétique
- signal analogique/digital,
- tension/courant,
- etc...

Certains couples (P, A) sont totalement incompatibles à ce niveau, et l'adaptateur aura, là encore, un rôle à jouer.

Il faut noter que les problèmes liés à l'adaptation des supports d'informations interviennent parfois sensiblement dans les choix organiques en vue de la réalisation de certains interfaces.

1.4.4. Résumé

D'après ce que nous venons de voir, l'adaptateur d'interfaces sera à même de remplir les tâches suivantes :

1. mémorisation temporaire de messages de P,
2. transcodage des messages de P,
3. suppression de certaines redondances au sein de ces messages,
4. adaptation des fonctions primaires et secondaires,
5. banalisation des formats de messages,
6. protection contre les erreurs de transmission,
7. adaptation des paramètres physiques,
8. établissement d'une liaison de données.

1.5. Décomposition fonctionnelle

D'après ce que nous venons de voir, l'interface va traiter un flux d'information en transit entre P et A. La structure des traitements peut se décomposer comme suit :

Réception

Transformation 1

Stockage

Transformation 2

Emission

Reprenons ces fonctions en détail en considérant que le flux d'information peut se décomposer en messages.

1.5.1. Fonction Réception

La fonction réception se charge de l'acquisition du flux d'informations provenant de P.

En nous limitant aux transmissions digitales, le flux d'informations sera constitué d'une séquence de bits qu'il s'agira en général de structurer en symboles (caractères) puis en messages.

Ce flux peut contenir, outre les symboles "utiles", des symboles redondants et des mécanismes de détection d'erreurs de transmission.

Les règles de structuration et de détection d'erreurs d'un parleur donné consitueront les spécifications logicielles S(P) et S(P,H) de ce parleur.

S(P,H) tiendra tout particulièrement compte du coupleur d'entrée utilisé. Les coupleurs intégrés actuels (USARTS, coupleurs parallèles, ...) se chargent en général de la structuration des bits en caractères ainsi que de la détection d'erreurs au niveau caractère (parité transversale).

S(P) fera toujours abstraction du coupleur d'entrée. Il fixera les règles d'obtention et de validation des messages.
La validation se basera sur des contraintes d'intégrité parmi lesquelles nous relèverons :

- parité longitudinale,
- règles syntaxiques,
- intervalle de temps maximum entre la réception de caractères successifs,
- code polynominal.

La structuration des caractères en messages suppose l'existence d'un mécanisme de synchronisation pouvant revêtir des formes diverses dont nous retiendrons :

- synchronisation par caractères de début et/ou de fin de message (ex : STX, ETX). Ces caractères seront uniquement employés en tant que délimiteurs.

- synchronisation syntaxique : une fenêtre se déplace sur le flux d'information et s'arrête sur une suite de symboles correspondant à la syntaxe d'une message de P.

- synchronisation temporelle : le premier caractère reçu après un long silence de transmission est interprété comme étant le premier caractère d'un message, le silence marquant la fin du message. Cette synchronisation ne sera utilisée que lorsque l'intervalle de temps séparant deux symboles d'un même message est très petit par rapport à l'intervalle séparant deux messages.

- synchronisation par signal électrique : la transition d'état d'un signal électrique spécialisé indique le début et/ou la fin d'une message.

En général, plus un flux d'information est redondant, plus aisée est la synchronisation.

1.5.2. Fonction "TST"

.... où "TST" signifie :

Transformation ---> Stockage ---> Transformation

La fonction "réception", examinée plus haut, a pour résultat de fournir soit un message de P, soit un indicateur d'erreur si au moins une des contraintes d'intégrité d'un message a été violée.

Nous avons vu que l'adaptateur pourrait jouer le rôle de régulateur de débit, de tampon et de filtrage de certains symboles. Ces rôles seront mis en oeuvre par la fonction TST.

P est non seulement un parleur invétéré, il parle souvent pour ne rien dire ... Les redondances sont souvent énormes à quelque niveau de structuration que l'on se trouve. Ces redondances sont inutiles à transmettre à A et rendent le stockage des données peu efficace. Elles seront éliminées dans une large mesure par la transformation pré-stockage.

L'expérience a cependant montré le danger d'effectuer des suppressions de redondances aux plus haut niveaux de structuration, là où la sémantique du message peut avoir un rôle important à jouer. Tant que l'adaptateur se borne à faire des transformations syntaxiques élémentaires, les spécifications fonctionnelles resteront simples et stables. Les transformations de plus haut niveau devraient dans la mesure du possible toujours être effectuées au sein du système auditeur, plus souple quant aux changements de spécifications.

Le stockage proprement dit doit être souvent important en volume, géré en "first-in" "first-out", et apte à traiter des messages de longueur variable. Des contraintes techniques ou économiques tendent à limiter l'espace-mémoire utilisable,aussi tout doit être fait pour rendre les informations les plus compactes possibles dans cet état transitoire où n'existent pas de critères de lisibilité par exemple.

La fonction de transformation "post-stockage" sera chargée de mettre en forme les données codées et stockées précédemment pour les rendre compréhensibles à A et pour banaliser éventuellement les formats fournis par un ensemble d'interfaces et ne consiste donc pas en une simple transformation inverse visant à obtenir tel quel le message reçu de P. C'est ici aussi que l'on effectuera, si nécessaire, des transcodages d'alphabets.

Un certain nombre de sous-fonctions sont sous-jacentes à la fonction "TST". Il faut en effet prévoir un mécanisme de synchronisation entre le vidage et le remplissage du tampon permettant d'indiquer le nombre de messages "stockés", et pour chaque message, sa longueur, son type, etc ...

De plus, on peut vouloir signifier à A bien autre chose qu'un message de P mais aussi, par exemple, que la capacité du tampon a été dépassée, que P a transmis un message incompréhensible ou que l'interface a été inopérant pendant un certain temps. Cette gestion d'erreurs ou d'avertissements à A devront également trouver leur place dans la fonction "TST" au sens large.

Notons que ces messages d'erreurs et d'avertissement seraient de peu d'intérêt pour les parleurs, ceux-ci ne disposant en général d'aucun moyen de recouvrement.

Les spécifications de la fonction TST dépendent tant de l'auditeur que du parleur et constitueront ainsi S(A,P).

1.5.3. Fonction Emission

Il s'agit ici de faire parvenir un message de P, passablement transformé, à A en s'assurant de sa bonne réception.

En passant en revue les différents problèmes liés aux logiciels de base des ordinateurs, nous avons vu que les différences de personnalité dans ce domaine peuvent être énormes; du plus simple au plus complexe.

Ce niveau de complexité, inexistant dans la majorité des cas chez les parleurs, nous amène à établir une analyse de la fonction émission sensiblement différente par rapport à la fonction réception.

On admet généralement <2> qu'une session de télé-communication peut se décomposer en 3 phases : établissement, transfert et terminaison.

Notons que nous nous plaçons dans un cadre très général dépassant les limites fixées par les protocoles de télécommunications classiques (mode de base, HDLC).

L'adaptateur d'interface, lorsqu'il contient des messages attendant d'être transmis, devra disposer des outils lui permettant d'exécuter des trois phases.

1. Etablissement

L'établissement peut être implicite (par le transfert immédiat des données, sans autre précaution) ou explicite. Dans ce dernier cas, ce sera soit l'adaptateur d'interface qui "appelera" A, soit A qui interrogera l'adaptateur d'interface. Ces deux modes de fonctionnement modifient sensiblement la stratégie et les outils à employer par l'adaptateur d'interface. Le premier mode (adaptateur d'interface = station primaire) exigera la présence de mécanismes d'horloge ("répéter toutes les x secondes si pas de réponse"), le second fera appel à un mécanisme d'interruption.

L'établissement d'une communication pourra parfois consister en un ensemble de commandes de "login", là où l'adaptateur d'interface se fait passer pour un utilisateur humain, sur une ligne banalisée de terminal. Ces commandes peuvent s'effectuer à l'aveugle, (transmission 1, délai, transmission 2, délai), si le système a des temps de réponse raisonnables et stables, ou sous forme de dialogue (transmission 1, attente et analyse de réponse, transmission 2).

Il faut noter ici que l'hypothèse simplificatrice de l'interface "uni-directionnel" évite de se poser les problèmes liés à la contention.

2. Transmission

La transmission proprement dite peut se décomposer en deux parties qui sont l'émission des données et l'interprétation d'une réponse éventuelle de A.

L'émission doit respecter les normes du langage compréhensible par A. La transformation éventuelle des données a été effectuée au préalable (TST) mais il faudra souvent y ajouter des éléments de synchronisation et de détection d'erreurs.

Pour des raisons d'efficacité, il peut être utile de regrouper plusieurs messages au sein d'une même transmission. Les critères de groupage seront alors une fonction du nombre de messages disponibles dans l'interface, de la longueur maximale d'un bloc de données pouvant être reçu par A et du délai s'étant écoulé depuis la dernière transmission.

Suite à l'émission, A pourra avoir un ensemble de réactions du type "bien compris", "répétez", "attendez" ou pas de réaction du tout ! Ces réactions devront être analysées et prises en compte. Notons ici encore qu'il sera souvent nécessaire de disposer d'une horloge afin de décréter la non-réception d'une réponse et d'éviter des attentes inutiles.

3. Terminaison

La phase de terminaison peut avoir lieu après chaque message, après un ensemble de n messages transmis, après un délai durant lequel l'interface n'a plus reçu de message de P, après réception d'une réponse donnée de A ou jamais. Elle a pour but de libérer une ligne et/ou un processus sur A ou de déclencher un traitement particulier des données déjà reçues par A.

La terminaison peut également être effectuée suite à un mauvais fonctionnement de la communication entre l'interface et A, dans l'espoir de rétablir une communication plus fructueuse.

Dans certains cas, la phase de terminaison peut précéder l'établissement d'une communication, ce qui permet à l'adaptateur d'interface de connaître précisément l'état de sa connexion avec A.

1.5.4. Fonctions auxiliaires

Sont reprises sous le terme générique de "fonctions auxiliaires" toutes les fonctions n'ayant pas de corrélation forte avec les mécanismes cités précédemment.

1. Vérification de la consistance du matériel

Un mécanisme de "self-test" du matériel de l'interface permet de détecter le mauvais fonctionnement de certains organes essentiels. La mémoire-programme, la mémoire données (tampon, variables de travail, ...), l'horloge, les organes d'entrée/sortie et le processeur lui-même peuvent faire l'objet de tels tests. Ces tests peuvent soit être effectués à l'initialisation de l'adaptateur d'interface, soit en temps que "chien de garde", c'est-à-dire en cours de fonctionnement et à intervalles réguliers.

Dans l'état actuel des choses, nos adaptateurs d'interfaces n'effectuent que des essais portant sur la mémoire et sur l'horloge à l'initialisation (mise sous tension).

Le choix d'un algorithme de détection d'erreurs devra se faire suivant les critères que sont la durée prise par le test et sa capacité de mettre des fautes en évidence.

2. Contact avec l'utilisateur

L'adaptateur d'interface étant un organe décentralisé, pouvant se trouver physiquement près de l'appareil "P", un contact avec l'utilisateur sera parfois souhaitable. Ces contacts pourront s'effectuer dans le sens interface vers utilisateur ou utilisateur vers interface.

a) Interface ----> Utilisateur

L'adaptateur d'interface peut ici dépasser son simple rôle de "boîte noire" en donnant certains signes de vie à l'utilisateur de P. Pour ce faire, il pourra par exemple afficher le nombre de messages en attente de transmission, afficher un code d'erreur et mettre en route une alarme sonore en cas de mauvais fonctionnement ou encore, imprimer, en clair les messages transmis vers A. Ce contact avec l'utilisateur pourra supprimer bon nombre de barrières psychologiques et fournira un bon outil de diagnostic en cas de difficultés de transmission.

b) Utilisateur ----> Interface

Dans certains cas, le parleur peut founir un ensemble de données incomplètes que l'auditeur ne peut traiter valablement.

Prenons l'exemple d'un appareil de mesure fournissant, sur un coupleur de sortie digital, les résultats des mesures effectuées sur un échantillon. Dans bon nombre de cas, l'auditeur devra connaître outre le résultat proprement dit, l'identification de l'échantillon concerné, qui n'est pas toujours fournie par le parleur. L'adaptateur d'interface peut combler cette lacune pourvu qu'il soit doté d'un dispositif de saisie de données auxiliaire (roues codeuses, clavier). Ce dispositif peut être considéré comme un second parleur travaillant en synchronisation avec le premier.

1.6. Analyse "particulière"

L'analyse "générale" du problème posé par la mise en oeuvre d'un adaptateur d' interface nous a permis de dégager une partie de l'univers des possibles. Quand il faut s'attaquer à l'analyse d'un cas particulier, c'est-à-dire d'un couple (P, A), la découverte est presque toujours au rendez-vous.

Si la structure des traitements peut être considérée comme relativement stable, la structure des données à traiter est essentiellement variable. Les "messages" peuvent se trouver dans 3 états différents au sein de l'interface :

1. état "P" : le message transmis par P
2. état "interne" : le message tel qu'il est stocké dans l'interface
3. état "A" : le message tel qu'il est transmis à A

Les spécifications liées à l'état "P" sont fixées par le constructeur de P; celles liées à l'état "A" par les contraintes du logiciel de base et du logiciel d'application de A; les spécifications de l'état interne seront liées à P et à A ainsi qu'à des contraintes de capacité de stockage.

Etats P et A :

Sans entrer dans des considérations trop particulières, voici un exemple de démarche d'analyse qui peut être suivie au niveau de l'état "P" et de l'état de "A" :

1. Niveau physique

- Quelle norme électrique est employée : en tension (V24, TTL) ou courant (20 ma).
- La transmission de données s'effectue-t-elle en série ou en parallèle ?
- Quel connecteur est employé, quelle est la signification de chaque broche ?
- Quel mode de transmission série, synchrone, asynchrone ?
- Quelle est la vitesse de transmission ?
- Si transmission synchrone, quel est le mécanisme de synchronisation ?
- Si une parité transversale est utilisée, de quel type est-elle ?
- Existe-t-il des signaux de contrôles spécialisés (contrôle de modem, horloge, synchronisation ..)
- Si un alphabet de caractères est utilisé, quel est-il (ASCII, EBCDIC, BAUDOT, ...)
- Le mode de fonctionnement est-il unidirectionnel, bidirectionnel, bidirectionnel à l'alternat ?
- .....

2. Niveau message

Considérons le message comme un vecteur de symboles de M de longueur L : M (L).

- L est-il une constante ? Sinon, quelles sont ses valeurs maximales et minimales ?
  Ex : L = 11 L ou $14 \leq L \leq 33$

- existe-t-il des symboles de synchronisation ?
  Ex : M(i) = STX si et seulement si i = 1
  M(i) = ETX si et seulement si i = L

- existe-t-il une règle de validation syntaxique du message ?

- existe-t-il une parité longitudinale, ou un autre mécanisme de détection d'erreurs ?

Etat interne

La démarche d'analyse relative à l'état interne tendra à répondre aux questions :

- Quels sont les symboles porteurs d'information au sein d'un message de P ?
- Quel est le nombre de symboles utiles différents ? Leurs fréquences relatives ?
- Les informations stockées auront-elles une représentation de longueur fixe ou variable ?

L'analyse de l'état interne se fera principalement au niveau organique où il s'agira de résoudre un compromis entre la densité de l'information stockée et la complexité de manipulation de cette information.

II ARCHITECTURE

II Architecture

Partant des fonctions à remplir par tout adaptateur d'interface, nous allons étudier ici comment vont s'assembler et coopérer les différentes briques composant le logiciel d'un adaptateur d'interface. Cette étude se veut générale à tout interface et ne traitera donc des algorithmes ou variables employées que d'une manière superficielle.

2.1. Décomposition en modules

Toute décomposition en modules d'une application met en oeuvre un certain nombre de critères, de règles d'obtention de cette découpe.

Dans le cas qui nous occupe, il faut mettre en évidence que nous n'avons pas affaire à un projet précis mais bien à un ensemble de projets ayant de nombreux traits en commun. Supposons que nous ayons déjà réalisé une connexion (P, A) et que notre nouveau projet consiste en une connexion (P', A). Il est évident que seules les spécifications dépendantes de P vont varier c'est-à-dire, pour reprendre la notation du chapitre précédent S(P) et S(P, A) qui vont devenir S(P') et S(P',A). Il est très fréquent que l'un des éléments du couple fasse déjà partie d'un couple existant.

De plus, il est essentiel lorsqu'il s'agit de faire face à la réalisation, mais aussi à la maintenance, de plusieurs dizaines de projets, d'établir une découpe en modules aussi stable que possible. Cette découpe réduit, dans une large mesure, le danger de redondances, facilite le travail de programmation et permet de retrouver aisément les différentes expressions d'un même concept dans l'ensemble des projets.

Les variables différenciant un projet d'un autre étant concrétisées par les spécifications de l'auditeur, du parleur et, dans une moindre mesure, de l'architecture matérielle, ce critère de dépendance fonctionnelle sera également celui de la décomposition en modules.

Ainsi, nous parlerons indifféremment des spécifications S(A) et du module S(A) qui lui correspond.

Dès lors, si une modifications de spécifications intervient pour un auditeur A, cette modification ne devra être portée qu'au module S(A) et, rarement, aux modules $S(P_1,A)$, $S(P_2,A)$, ... Rarement car ces modules travaillent sur une structure de données intermédiaire (état interne, voir 1.6) dépendant fort peut de A.

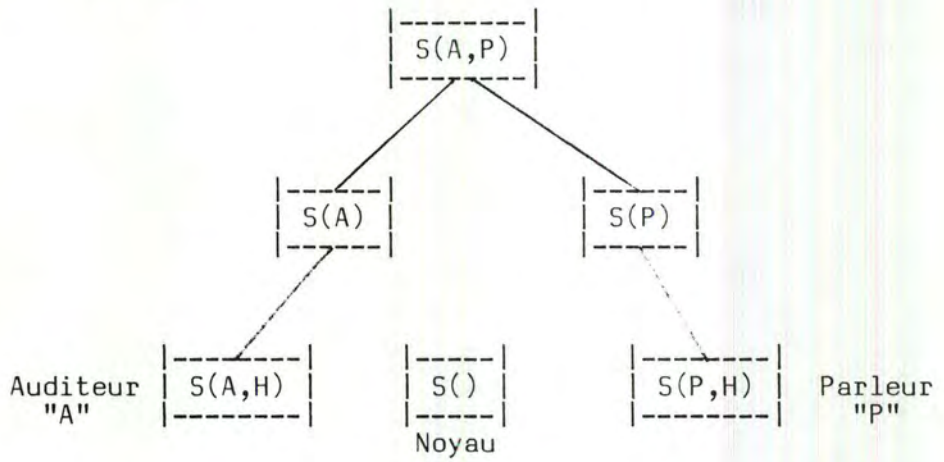
Un projet particulier se décomposera donc en 5 modules :
S(A), S(A,H), S(A,P), S(P,H), S(P)
parmi lesquels seul S(A,P) est spécifique à ce projet.

En outre, un certain nombre de fonctions, très générales sont suffisamment indépendantes fonctionnellement de A, P et H pour tenir une place à part, soit S(). Le module S() regroupant un ensemble de fonctions d'assez bas niveau sera appelé le "noyau", par analogie aux fonctions de base des systèmes d'exploitation. Le contenu de ce noyau sera examiné en détail dans un paragraphe particulier.

La décomposition fonctionnelle établie au chapitre précédent "colle" bien à la décomposition modulaire, ainsi :

S(A,H) + S(A) = fonction réception
S(A,P) = fonction "TST"
S(P,H) + S(P) = fonction émission
S() = noyau, fonctions diverses

Le schéma II.1 nous montre, de manière très générale, comment s'opère l'échange d'information entre les différents modules.



II.1

Notons que les informations transitent presque toujours en tant que caractères entre S(P,H) et S(P) ainsi qu'entre S(A,H) et S(A), alors qu'elles prennent la forme de messages entre S(P) et S(A,P) ainsi qu'entre S(A,P) et S(A).

2.2. Le noyau

Le noyau rassemble, comme le fait un système d'exploitation, un grand nombre d'outils d'intérêt général faisant part de nombreux logiciels d'adaptation d'interfaces. Ce n'est cependant pas un "vrai" système d'exploitation, en effet, il laisse une large part de responsabilités entre les mains du programmeur. Le noyau offre une gamme d'outils standard optimisés, évitant les redondances d'écritures de sous-routines. Voici les fonctions qu'il offre au programmeur :

- initialisation,
- protection de zones critiques,
- gestion d'horloges,
- gestion de processus,
- gestion du tampon,
- compteur général,
- divers.

2.2.1. Fonction d'initialisation

Au démarrage de l'adaptateur d'interface, un certain nombre d'opérations doivent être effectuées, citons entre autres :

- tests mémoire,
- tests périphérie,
- initialisation du système d'interruptions,
- initialisation des variables de travail.

pour ensuite donner le contrôle au programme d'application proprement dit. Toutes ces opérations sont effectuées par la fonction d'initialisation.

La découpe en modules suivant des critères fonctionnels implique l'existence d'initialisations propres à ces modules qui ne peuvent donc être intégrées telles quelles dans le noyau.

Prenons l'exemple d'un auditeur nécessitant la mise en oeuvre d'un organe d'entrée-sortie série asynchrone. L'initialisation de cet organe dépend fonctionnellement de S(A,H) et ne peut donc être généralisée au niveau du noyau.

Pour résoudre ce problème, il a été nécessaire de créer une déclaration de procédure d'initialisation. Le programmeur peut ainsi fixer les conditions initiales au sein d'un module donné, la fonction d'initialisation se chargeant de l'exécution de toutes les procédures ainsi déclarées. Le mécanisme permettant ces déclarations est inclus dans le noyau.

Notons cependant que l'ordre dans lequel les initialisations seront effectuées à l'exécution est inconnu du programmeur.

2.2.2. Protection de zones critiques

Tout programme "temps réel" travaillant avec des interruptions ou en multi-processeurs, connaît des problèmes de protection de zones critiques. Cette protection se résume souvent en un masquage suivi d'un dé-masquage du système d'interruptions. Cette opération, pour certains microprocesseurs, n'est cependant pas toujours triviale.

En effet, il faut, lorsque l'on masque les interruptions, pouvoir se souvenir de l'état du système d'interruptions avant le masquage. Le noyau offre deux primitives au programmeur : prot() et unprot() qui garantissent l'intégrité du système d'interruption.

2.2.3. Fonction horloge

La présence d'une horloge est essentielle dans tout système "temps réel" afin d'éviter des attentes infinies ou trop prolongées sur certains événements, ou simplement pour introduire des délais. Le noyau offre un tel mécanisme d'horloge, gérant les interruptions d'une horloge hardware externe, et ayant une résolution de 1/100ème de seconde.

Le programmeur peut déclarer jusqu'à 128 horloges qui sont identifiées par le nom de la procédure à exécuter après un délai programmé. Notons que la création d'une horloge se fait statiquement, à l'assemblage suite à la présence d'une déclaration du type, timer(procédure = timer_name). Le programmeur peut utiliser trois type de primitives qui permettent de démarrer une horloge, de l'arrêter ou de la relancer, soit :

```
runtim(timer_name, delay)
stoptim(timer_name)
contim(timer_name).
```

Exemple :

```
timer (teatime)----> déclaration d'horloge
teatime :      ----> routine de gestion d'horloge
... what to do at 5 o'clock

runtim(teatime, 100) ----> "teatime" sera appelé 100
                           secondes après l'exécution de
                           runtim.
```

Une certaine discipline d'écriture doit être suivie pour les routines de gestion d'horloge qui doivent avoir un temps d'exécution minime.

2.2.4. Gestion de processus

L'adaptation d'interfaces exige l'exécution quasi simultanée de plusieurs tâches ou processus. Ainsi, la réception d'un caractère provenant du parleur peut avoir lieu pendant la transmission d'un caractère vers l'auditeur. Il sera donc nécessaire de disposer d'un mécanisme permettant d'implémenter ce parallélisme d'actions étant donné que nous ne disposons que d'un seul processeur. Le gérant des processus est un de ces mécanismes, fréquemment utilisé dans les systèmes d'exploitation. Signalons cependant que son emploi ne s'impose pas toujours. En effet, dans bon nombre de cas, les gestionnaires d'interruptions peuvent parfaitement assurer une bonne découpe en processus de l'application. Cependant, lorsque la complexité devient plus grande, un véritable gestionnaire de processus s'impose.

Le noyau contient facultativement un gérant de processus appelé "UPROM" (pour Unina Processes Manager). Uprom offre un mécanisme de déclaration statique de processus, c'est-à-dire résolue à l'assemblage. Pour des raisons d'efficacité, il n'implémente pas de mécanisme de création ou de suppression dynamique de processus. Les processus qu'il gére auront donc une structure de boucle infinie. Le partage du processeur s'effectue sur base d'appels à des primitives d'attente d'événements effectués par les processus à des intervalles de temps relativement courts. Le gérant des processus ne peut donc pas, de sa propre initiative, interrompre un processus en cours d'exécution.

Cette contrainte exige, de la part du programmeur, une discipline d'écriture des processus qui s'avère cependant peu contraignante dans la pratique, tout en évitant la complexité des systèmes en temps partagé.

"UPROM" traite deux types d'objets liés aux processus qui sont, d'une part le contexte (pile) qu'il sauve et restaure et, d'autre part, un état qui peut prendre les valeurs suivantes :

- actif (au plùs un processus à la fois),
- en attente d'événement,
- en attente du processeur, "processeur prêt" étant un événement particulier.

Initialement, tous les processus sont en attente du processeur.

Les gérants de processus font, de près ou de loin, toujours appel à la notion d'événement. UPROM ne fait pas exception à la règle et utilise les événements comme moyen de synchronisation.
L'événement, pour être connu de UPROM, doit lui aussi être déclaré, sous la forme event(event_name). L'événement, à quelques différences près, rejoint la notion de "semaphore". Comme le sémaphore, il est l'ensemble d'un compteur/indicateur d'état et d'une file d'attente.

Les états possibles d'un événement sont :

- l'événement a eu lieu, sans encore être "utilisé"
- l'événement est attendu (par au moins un processus)
- l'événement n'a pas eu lieu et n'est pas attendu.

L'algorithme de file d'attente est du type : premier arrivé, premier servi, sans autre notion de priorité.

En dehors des déclarations proprement dites, UPROM offre deux primitives wait(event_name) et occurs(event_name), sensiblement équivalentes aux primitives P et V des sémaphores <3>.
La primitive wait, attente d'événement, fonctionne comme suit :

- si l'événement a eu lieu, on dira que "l'événement n'a pas eu lieu" et le processus actif reste actif,

- sinon, le processus actif passe en attente de l'événement et UPROM sélectionne un nouveau processus susceptible d'être activé.

Notons ici une différence importante par rapport aux sémaphores : l'événement n'a une notion de comptage que pour le nombre de processus en attente, et non pas du nombre d'occurrences de l'événement (on ne sait pas répondre à la question : combien d'occurrences de l'événement "non attendu" ont eu lieu). Les sémaphores présentent en effet des effets de bords très désagréables dans le genre d'applications qui nous concernent. Ces effets de bords résident dans la différence existant entre les concepts de ressources et d'événements.

Un cas particulier est wait(processor), qui a pour effet de rendre la main à UPROM, afin de donner une chance à un autre processus d'être sélectionné, sans rester en attente d'un événement particulier.

La primitive occurs signale une occurrence d'un événement et fonctionne comme suit :

- si l'événement a déjà eu lieu, ne rien faire (!)
- si l'événement n'a pas déjà eu lieu et n'est pas attendu, signaler qu'il a eu lieu,
- si l'événement est attendu, sélectionner le premier processus dans sa file d'attente et le mettre en attente du processeur.

L'emploi de la primitive occurs sera particulièrement fréquent dans les routines de gestion d'interruptions afin par exemple de signaler la fin d'une opération d'entrée/sortie.

Le diagramme II.2 nous montre les 3 états dans lesquels peuvent se trouver les différents processus ainsi que les transitions possibles.

Voici une brève description des 4 transitions d'états :

1. Le processus a été sélectionné lors d'une occurence de l'événement sur lequel il restait en attente.
2. Le processus a été sélectionné dans la file d'attente du processeur.
3. Le processus a exécuté une primitive wait (processor)
4. Le processus a exécuté une primitive wait (event)

2.2.5. Gestion du tampon

En procédant à l'analyse fonctionnelle des adaptateurs d'interfaces nous avons mis en évidence la nécessité de fournir la possibilité de stocker un grand nombre de messages en attente de transmission. Cette fonction de mémorisation est suffisamment générale pour prendre place dans le noyau où elle portera le nom de "gestion du tampon". Cette gestion devra fournir des primitives de haut niveau (lire, écrire) très efficaces quant au rendement de l'utilisation de la mémoire. La taille de cette mémoire est en effet limitée par des facteurs tant technologiques (espace d'adressage, consommation) qu'économiques; les tailles standards se situent entre 2000 et 20000 bytes. Le tampon est organisé sous forme circulaire pour réaliser un algorithme "FIFO". Son unité de traitement est le demi-byte. En effet, une bonne partie des informations à traiter sont présentées sous forme décimale qui permet un codage sur 4 bits. D'autre part, et toujours dans le but de rendre les informations les plus compactes possibles, une petite unité de mémorisation permet d'employer une stratégie s'inspirant des codes d'Huffman qui peut se résumer comme suit **<7>** "plus la fréquence d'un symbole est grande, plus sa représentation sera compacte".

Un ensemble de 4 primitives permet de lire ou d'écrire un byte ou un demi-byte dans le tampon. Ces primitives ne sont pas protégées contre les dépassements de capacité du tampon, tant par le bas (underflow) que par le haut (overflow). Pour prévenir le dépassement par le haut, une primitive permet de répondre à la question :"y a-t-il au moins un espace libre de n demi-bytes dans le tampon?"

La prévention de dépassement par le bas exige l'établissement d'un protocole, d'un mécanisme de synchronisation, entre les fonctions de vidage et de remplissage du tampon. Cette synchronisation utilisera toujours un compteur de messages comme décrit au paragraphe suivant.

Dans certains cas, le parleur fournira des messages particulièrement longs dont les contraintes d'intégrité ne peuvent être vérifiées qu'après réception complète. Ce genre de situation implique l'existence d'une zone de mémorisation temporaire des messages. Afin d'éviter le gaspillage de mémoire ainsi provoqué, le gérant du tampon offre la possibilité de sauver et de restaurer le contexte du tampon, autorisant ainsi le stockage temporaire de données pouvant être suivi soit d'un effacement, soit d'une confirmation.

Le gérant du tampon n'effectue aucun arbitrage d'accès son bon fonctionnement n'étant garanti que lors de la présence d'un seul producteur et d'un seul consommateur. Cette restriction peut aisément être levée à l'aide du gérant des processus et des primitives de protection des sections critiques <1>.

2.2.6. Compteur général

Le gérant du tampon ne disposant d'aucune notion de messages, mais bien seulement de caractères, il sera nécessaire de lui adjoindre un mécanisme permettant une synchronisation entre le processus producteur de messages et le processus consommateur de ces messages. Ce sera le but poursuivi par la fonction de comptage qui sera donc fonctionnellement très dépendante de la gestion du tampon.

Le compteur général doit donc aussi offrir une grande sécurité d'emploi quant à la protection des sections critiques.

Initialisée à 0, sa valeur peut être incrémentée ou décrémentée par l'appel de deux primitives qui ont également pour résultat d'afficher le nouveau contenu du compteur sur un dispositif d'affichage, périphérique du micro-ordinateur/adaptateur. Cet affichage peut ainsi servir à donner vie à l'adaptateur en montrant le rythme de remplissage et de vidage des messages.

Une primitive indique si le compteur est à zéro ou non; elle pourra être utilisée comme critère pour l'établissement d'une transmission de données.

Une dernière primitive est utile lorsque l'on veut savoir s'il y a au moins n messages en attente de transmission afin de grouper des messages en un seul bloc.

2.2.7. Fonctions diverses

Sont regroupées sous ce paragraphe, un ensemble de primitives intéressantes à divers titres :

- routine de conversion de binaire en ASCII (hexadécimal),
- gestion de l'affichage comprenant une routine de conversion de binaire en code 7 segments,
- gestion des erreurs, aidant à l'affichage des codes d'erreur et déclenchant pour un temps limité une alarme sonore,
- déclarations de macro-instructions d'intérêt général.

Le noyau utilise en outre une table décrivant la configuration matérielle de l'interface (adresses des périphériques d'entrée/sortie, configuration mémoire, fréquence d'horloge, etc ...) ainsi qu'un ensemble d'informations utiles au pilotage des périphériques, aux conversions de codes, etc ...

2.3. Décomposition en processus

Les processus tiennent une part importante dans les logiciels en temps réel. Après avoir examiné ce que le noyau peut nous offrir comme outils (2.2.4) en ce domaine, nous allons examiner plus en détail les choix offerts lors de la réalisation d'un logiciel d'adaptateur d'interface.

La décomposition en processus, dans le cadre limité de l'interface uni-directionnel est relativement simple. En gros, deux processus asynchrones suffisent :

- l'un en entrée assure les fonctions de réception, de transformation et de stockage des données,

- l'autre en sortie assure les fonctions de déstockage, transformation et émission.

Ajoutons à cela un pseudo-processus constitué par l'horloge. Notons ici qu'il n'existe pas de relation simple entre les modules et les processus. En effet, la découpe en modules selon les critères de dépendances fonctionnelles ne favorise pas une découpe en processus particulière.L'implémentation de cette découpe en processus peut être réalisée de différentes manières, nous en retiendrons deux :

- les processus sont situés à des niveaux d'interruption différents,

- les processus sont tous situés au niveau de base.

Dans le premier cas, un seul processus se trouve au niveau de base, les autres étant des gérants d'interruption.

Pour les applications simples et n'ayant que de faibles débits de données à traiter, cette méthode simplifie la réalisation du logiciel. Le processus de réception qui, nous l'avons signalé, doit être prioritaire, s'effectue en réponse à une interruption générée par la réception d'un caractère. Le processus d'émission, peu prioritaire, utilise tout le temps laissé libre pour boucler en attente de message ou gérer l'émission d'un message. Le gros inconvénient de cette méthode réside dans le fait qu'une gestion d'interruption peut nécessiter un temps très important (à la réception du dernier caractère d'un message par exemple) de manière ponctuelle.

Dans le deuxième cas cet inconvénient, pouvant causer la non-réception correcte d'un message, est résolu par l'utilisation d'un gérant de processus tel qu'UPROM. L'emploi du processeur dans le temps peut être rendu beaucoup plus "lisse", les gérants d'interruptions n'ayant à effectuer que des opérations élémentaires. Dans ce cas, les différents processus se trouvent sur pied d'égalité et le bouclage, que l'on trouve dans tout système temps réel, ne se situe plus dans la routine d'émission ("y a-t-il quelque chose à émettre ?") mais bien dans le gérant de processus ("y a-t-il un processus prêt ?").

L'emploi d'UPROM rend cependant plus complexe la réalisation du logiciel. La succession des événements et des actions est beaucoup moins bien définie, les précautions à prendre sont plus nombreuses. Ces précautions consisteront souvent en l'utilisation de compteurs, de variables de travail, de tampons, etc ...

Que l'on choisisse l'une ou l'autre méthode, le problème des zones et variables critiques reste quasiment le même. Rappelons que certaines variables sont dites "critiques" lorsqu'elles peuvent être accédées par plus d'un processus et que l'accès quasi simultané peut causer des ambiguités. Le sujet est traité par ailleurs en long et en large, dans tous les ouvrages traitant des systèmes d'exploitation <1>, <3>. Dans l'application qui nous occupe, la protection des variables critiques est une opération souvent coûteuse qui ne doit pas être appliquée systématiquement. Une étude précise de chaque variable ou zone critique doit être effectuée afin de mettre en lumière les effets de bord causés par des accès quasi-simultanés. Ces effets sont le plus souvent acceptables au niveau du programme d'application (qui utilise d'ailleurs presque exclusivement des variables locales !).

III. REALISATION DES ADAPTATEURS D'INTERFACES

III. Réalisation des adaptateurs d'interfaces

Ce chapitre a pour but de décrire les différents outils et méthodes mis en oeuvre dans la réalisation d'un logiciel d'adaptation d'interface. Les outils y seront cités dans l'ordre de leur usage pendant la réalisation, la vie d'un projet déterminé, de l'analyse à la phase de tests en passant par la programmation proprement dite.

3.1. Analyse du fonctionnement de P et de A

Les spécifications des interfaces de communication des ordinateurs et des autres appareils susceptibles d'être interconnectés doivent, en théorie, être fournies par les constructeurs. Entre la théorie et la pratique, il existe dans ce domaine une différence très importante. Les descriptions techniques (quand elles existent !) sont toujours inutilisables telles quelles, du moins lorsque l'on s'éloigne de procédures standard (BSC, HDLC, IEEE 488).

L'expérience nous a montré qu'il est toujours nécessaire d'effectuer des essais "sur le terrain" afin de se faire une idée précise du mode de fonctionnement d'un appareillage donné. Voilà de quoi donner des sueurs froides aux informaticiens ne concevant leur travail que sur base de spécifications précises et rigoureuses ! Heureusement, la complexité des interfaces les moins bien spécifiés est assez faible et le bon sens peut être utilisé pour mettre en lumière tel ou tel point mal défini.

Afin de simplifier l'exposé, nous ne nous intéresserons qu'à l'analyse du fonctionnement d'un "parleur".

3.1.1. Analyse de l'interface physique

L'interface physique, a priori plus facile à analyser que les interfaces de plus haut niveau, présente cependant une grande variété de possibilités qui est à la base de nombreux problèmes. Ces problèmes étant plus souvent du ressort de l'électronicien que de l'informaticien, nous nous bornerons à en citer quelques un.

Le constructeur du parleur fera souvent référence à la norme V24 du CCITT. Il omettra cependant d'indiquer si son appareil doit être considéré comme un équipement de terminaison de circuit de données ou comme un équipement terminal de traitement de données..

Dans d'autres cas, il sera fait usage d'une boucle de courant (20 mA), sans préciser si le parleur est actif ou passif dans cette boucle. De plus, l'anarchie la plus complète règne dans le domaine des connecteurs mécaniques.

L'analyse de cet interface physique s'effectue souvent selon une démarche d'essais et erreurs, qui peut être très fastidieuse.

Notons que la multiplication des adaptateurs d'interfaces nécessite la création d'un outil de documentation concernant les différents interfaces physiques, encore inexistant à l'heure actuelle.

3.1.2. Analyse de la structure du flux d'informations

Lorsque l'on désire interpréter ou transformer le flux d'informations provenant d'un parleur, il est essentiel de connaître les règles de structuration que ce flux respecte. Il s'agira de répondre à un ensemble de questions parmi lesquelles nous relèverons :

- Le flux d'information étant décomposé en messages, existe-t-il plusieurs types différents de messages? (message d'en-tête, de commentaire, d'alarme, etc ...)

- Les messages étant composés de champs distincts, quels sont ces champs (champ de résultat, d'identification, ...) ?
  Quelles en sont les caractéristiques (position, longueur, présence facultative ou obligatoire, ...) ?

- L'objet de base étant le caractère, comment celui-ci est-il codé (ASCII, EBCDIC, ...) ? Existe-t-il des caractères spéciaux, non imprimables ?

Le constructeur du parleur ne fournira pas toujours des réponses complètes et exactes à ces questions pourtant fondamentales.

Il s'agira donc systématiquement de fixer les spécifications réelles du parleur en validant, refusant ou complétant les spécifications du constructeur. Notons ici que cette tâche n'est réalisable que lorsque le niveau de complexité du parleur est faible, ce qui est généralement le cas.

La mise en évidence des caractéristiques du flux d'information se fera en deux étapes :

1° Obtention d'un échantillon le plus exhaustif possible du flux,

2° Interprétation de cet échantillon.

L'obtention de l'échantillon se fera "in situ", elle consistera en la collecte du flux d'information provenant du parleur pendant un laps de temps garantissant l'obtention d'un échantillonnage quasiment exhaustif. Les données ainsi recueillies seront soit imprimées, soit stockées sur une mémoire de masse (ex : disquette), cette dernière méthode étant plus avantageuse car elle permet le transfert aisé des données sur un système informatique.

Lors de l'échantillonage, il sera souvent fait appel à des adaptateurs d'interfaces particuliers assurant la mise en évidence de caractères non imprimables et la compatibilité des interfaces physiques. Ces adaptateurs (appelés "transp" pour transparents) suivent la structure modulaire standard. Ils ne font qu'un nombre d'hypothèses très réduit quant au parleur afin de pouvoir être utilisés dans le plus grand nombre de situations possible. Les couples (P,A) qu'ils réalisent seront ici (transp, disquette) ou (transp, imprimante).

L'interprétation de l'échantillon consiste en la mise en évidence des caractéristiques du flux d'informations. Cette opération, souvent fastidieuse, peut être facilitée par l'emploi de programmes utilitaires. Ces programmes effectueront, par exemple le transcodage d'une représentation hexadécimale en caractères ASCII et vice versa, ajouteront ou supprimeront les parités transversales, etc... D'autres utilitaires donneront une liste des caractères utilisés, leurs fréquences relatives, leurs positions dans des messages de longueur fixe. La portée de ces programmes reste cependant limitée à la mise en évidence de caractéristiques syntaxiques de très bas niveau tout en constituant un bon outil de documentation.

Enfin, signalons que l'échantillon recueilli lors de la phase d'analyse sera fréquemment utilisé comme base pour des jeux de tests dont il sera question plus loin.

3.2. Réalisation du logiciel d'adaptation

La phase de programmation dont il est question ici s'étend de l'écriture du programme-source d'un adaptateur d'interface, au chargement dans la mémoire d'un micro-ordinateur de ce programme sous forme exécutable. Cela suppose le parcours de plusieurs étapes suivant une méthode générale de travail dont nous étudierons les principales caractéristiques. Certaines étapes seront marquées par l'usage d'outils spécialisés (macro-processeur, moniteur) qui retiendront plus particulièrement notre attention.

3.2.1. Les étapes de la réalisation

Chaque nouvelle application nécessite l'écriture, au moyen d'un éditeur de texte, d'un ou de plusieurs modules "S" auxquels nous avons déjà fait référence.

Ces modules sont écrits en langage assembleur, plutôt qu'en un langage de haut niveau essentiellement pour garantir une plus grande vitesse d'exécution. Les inconvénients inhérents au langage assembleur (manque de clarté, manque de portabilité, ...) sont partiellement compensés par l'emploi de macro-instructions.

Lorsque nous disposons de tous les modules-sources d'une application donnée, ceux-ci sont concaténés en un seul programme-source. La concaténation des sources a été préférée à l'édition des liens portant sur les modules-objets pour plusieurs raisons, dont nous retiendrons :

- la taille du programme-source ainsi obtenue reste relativement faible (de 30 000 à 50 000 caractères),

- les modules, découpés suivant des critères de dépendance fonctionnelle, peuvent communiquer entre eux par des mécanismes de haut niveau (souvent des macro-instructions) lors de l'élaboration du programme exécutable. L'éditeur de liens interviendrait ainsi trop tard dans cette élaboration,

- la présence d'une seule source facilite la maintenance d'un projet précis. En effet, elle constitue un catalogue implicite des différents modules utilisés et ... de leur version,

- l'éditeur de liens doit tenir compte de l'existence de mémoires n'autorisant pas l'écriture; ce n'est pas toujours le cas.

La concaténation est suivie de deux étapes distinctes qui consistent d'une part en la traduction des macro-instructions, d'autre part en l'assemblage proprement dit. Nous avons en effet préféré utiliser un macro-processeur général plutôt qu'un macro-assembleur intégrant les deux étapes. Le macro-processeur est plus "général" dans le sens qu'il ne fait quasiment aucune hypothèse sur le texte qu'il manipule.

L'assembleur produit un code-objet qu'il s'agit ensuite de charger dans la mémoire d'un micro-ordinateur. Notons ici que toutes les étapes précédentes peuvent demander un travail considérable qui ne peut être effectué que sur un mini-ordinateur, ici un PDP 11. Le programme-objet doit donc être transféré de ce mini-ordinateur soit vers une mémoire non-volatile, par exemple une EPROM (Erasable & Programmable Read Only Memory), soit vers la mémoire volatile du micro-ordinateur chargé de l'adaptation d'interface. Cette dernière possibilité ne sera utilisée que lors de la phase de test sur un micro-ordinateur particulier quant à son architecture matérielle. Ce micro-ordinateur sera également doté d'un logiciel "minimum", le moniteur, que nous étudierons plus loin.

3.2.2. Les macro-instructions

L'emploi des macro-instructions a constitué un outil privilégié lors de la réalisation des programmes d'adaptation d'interface et cela pour des raisons fort différentes :

1) L'assembleur utilisé doit être qualifié de "minimal". Il n'offre pas certaines possibilités cependant très intéressantes, telles que l'assemblage conditionnel et les manipulations de chaînes de caractères. L'emploi de macro-instructions permet de combler cette lacune.

2) Le jeu d'instructions du micro-processeur utilisé, le 8085 d'INTEL, est réduit. Les macro-instructions permettent de fournir un jeu d'instructions plus étendu par la combinaison d'instructions de base. Ces opérations seront entre autres :

   - sauvetage, restauration de plusieurs registres,
   - manipulation de demi-bytes,
   - protection de zones critiques,
   - masquage/démasquage sélectif d'interruptions.

3) Le micro-processeur dispose de deux types de mémoire :

- la mémoire ROM, non volatile, contenant le programme proprement dit ainsi que des tables de constantes,

- la mémoire RAM, volatile mais autorisant les écritures, contenant toutes les variables utilisées par le programme.

Il est donc nécessaire, lors de l'écriture des programmes, de distinguer ces deux types de mémoire. L'assembleur étant incapable d'effectuer cette distinction, un mécanisme de déclaration de variables a été réalisé à l'aide de macro-instructions. Ce mécanisme redéfinit récursivement une table des variables à chaque nouvelle déclaration.

4) L'assembleur ne peut manipuler que des objets de faible niveau de complexité. Les seules déclarations qu'il autorise consistent en l'attribution d'un nom à une valeur, par exemple une adresse (equates). L'emploi des macro-instructions permet de réaliser des mécanismes de déclarations d'objets de plus haut niveau. Citons entre autres :

- déclarations d'horloges et de leurs procédures de gestion,
- déclarations de processus et d'événements,
- déclarations de procédures d'initialisation.

L'emploi de ces déclarations permet de rendre fort général un grand nombre de procédures, sans tomber dans le piège des nombreux tests conditionnels lors de l'exécution du programme d'adaptation.

5) Certains modules "S" contiennent ponctuellement des paramètres qui varient d'une application à l'autre. Plutôt que de créer des copies de ces modules répondant à chaque valeur possible de ces paramètres, il est préférable d'actualiser leur valeur de manière interactive. Cette actualisation est rendue possible par la création d'une macro-instruction "interactive" dont le texte de remplacement est introduit via un terminal lors de l'évaluation par le macro-processeur. Cette possibilité a nécessité certaines modifications du macro-processeur proprement dit.

6) Certaines informations fort répétitives ou peut importantes fonctionnellement peuvent être masquées par l'emploi de macro-instructions. Ainsi, la quasi-totalité des appels au noyau s'effectuent par l'appel de macros. Le programmeur n'a pas à se soucier du mode de passage des paramètres et du sauvetage des registres par exemple. La véritable fonction d'une procédure peut être mise en valeur par rapport à des mécanismes de très bas niveau.

En résumé, les macro-instructions permettent de résoudre les problèmes suivants :

- réponses aux lacunes de l'assembleur,
- réponses aux lacunes du jeu d'instructions du micro-processeur,
- déclaration de variables,
- déclaration d'objets de haut niveau (horloges, processus, événements, ...)
- actualisation de paramètres fortement variables,
- masquage des informations inutiles,
- brièveté de l'écriture.

Le macro-processeur utilisé a été conçu pour être d'usage très général; il ne fait que peu d'hypothèses concernant les textes à traiter. Son fonctionnement, proche de celui des langages fonctionnels tels que LISP, est explicité dans **<4>**. Certaines fonctions pré-définies ("built-in macros") lui ont été ajoutées pour les besoins de notre application, le plus souvent pour des motifs d'efficacité.

3.2.3. Le moniteur "U-85"

Le programme objet, élaboré sur un mini-ordinateur, doit être transféré dans la mémoire d'un micro-ordinateur soit pour effectuer des tests, soit pour constituer le produit final. Cette opération de transfert et de chargement est effectuée à l'aide d'un poste de travail comprenant un micro-ordinateur dont l'architecture matérielle, à deux exceptions près, est identique à celle des adaptateurs d'interfaces. Ces deux exceptions sont :

- la présence d'un programmeur de mémoires EPROM,
- l'existence d'un mécanisme de "jump on reset" permettant, à l'initialisation du micro-processeur, de commencer l'exécution des instructions à une adresse pré-déterminée.

Ce micro-ordinateur dispose de deux coupleurs d'entrée/sortie; le premier coupleur établit une connexion avec le mini-ordinateur, le second avec un terminal de contrôle. Il dispose d'un logiciel de base, que nous appellerons le moniteur "U-85", permettant d'exécuter quelques vingt commandes différentes que nous pouvons regrouper comme suit :

- commandes de connexion au système-hôte,
- commandes de programmation d'EPROM, avec vérification du contenu ou de l'effacement,
- contrôles d'exécution de programmes avec insertion de points d'arrêts, permettant d'exécuter des programmes en temps réel ou en pas à pas,
- édition de mémoire en hexadécimal et en ASCII; désassemblage,
- édition des registres du processeur,
- contrôle des portes d'entrée/sortie.

La commande de connexion au système-hôte permet de rendre le micro-ordinateur complètement transparent entre le terminal de contrôle et l'hôte, ou encore, d'être simultanément transparent et de charger sa mémoire de messages particuliers émis par le système-hôte. C'est bien sûr par ce biais que seront transmis et chargés les programmes-objets.

Ces programmes, une fois chargés, sont ensuite copiés en mémoire EPROM ou testés à l'aide de simulateurs dont nous parlerons plus loin.

Le lecteur trouvera de plus amples détails concernant le moniteur "U-85" dans les annexes à ce mémoire.

Notons encore que le double couplage réalisé, à savoir : hôte/micro et micro/terminal de contrôle, connaît une alternative intéressante qui serait constituée des couplages hôte/terminal et hôte/micro. Cette alternative, quoique moins performante et plus complexe à réaliser, permet de libérer un des coupleurs d'entrée/sortie du micro-ordinateur, ce qui s'avère souvent utile dans la pratique.

3.3. Les outils de tests

Les tests représentent une part importante dans la réalisation d'un logiciel d'adaptation. Deux types d'outils sont utilisés à cette fin : d'une part les simulateurs et d'autre part les outils permettant d'interpréter certains résultats de simulation, nous les nommerons "outils de visualisation".

3.3.1. Les simulateurs

La validation du fonctionnement des interfaces se fait par simulations intensives. Des simulateurs sont chargés d'imiter le travail de différents "parleurs" et "auditeurs". Ces simulateurs sont soit réalisés sur un micro-ordinateur ayant la même configuration matérielle qu'un interface standard, soit sur le PDP-11.

Ainsi, l'on a écrit des émulateurs de système d'exploitation opérant en parallèle avec un simulateur d'auto-analyseur résidant tous les deux sur le mini-ordinateur .

En règle générale, à chaque "parleur" correspond un programme de simulation comprenant un noyau réduit ainsi qu'un module reprenant une partie des spécifications S(P) et S(P, H).

Il n'y a évidemment pas de liens fonctionnels avec un auditeur donné. Le simulateur dispose d'un ensemble de "messages types" répondant à la syntaxe utilisée par le parleur. Cet échantillon de messages doit être le plus exhaustif possible et provient souvent d'une collecte de données effectuées au préalable (voir 3.1).

3.3.2. Outils de visualisation

Il est souvent utile de vérifier, en simulation ou sur le site définitif, le bon fonctionnement des interfaces.

Cela concerne tout particulièrement, vu la complexité des situations possibles, le dialogue avec l'auditeur.

Il a déjà été question (3.1.) d'outils de visualisation de monologues. Un autre programme de micro-ordinateur, le "bi-scope", permet de visualiser le déroulement d'une communication bi-directionnelle en fournissant les caractères émis par deux sources de manières à pouvoir les distinguer aisément. Les problèmes d'incommunicabilité peuvent ainsi être mis en évidence.

Les outils de visualisation, d'usage assez général dans les télécommunications, existent en grand nombre dans le commerce, mais leur prix est exorbitant. C'est le barrage financier également qui empêche actuellement l'emploi d'outils de mise au point de micro-ordinateurs tels que les émulateurs en temps réel dont l'aide serait souvent appréciable.

CONCLUSION

Conclusion

Malgré une volonté certaine de normalisation dans les interfaces de communication, les adaptateurs d'interfaces semblent avoir de beaux jours devant eux, sinon les plus beaux. Leurs réalisation ne peut pas se baser, par définition, sur une politique du type "prêt à porter", mais bien du type "sur mesure" donc coûteuse en logiciel. Il est nécessaire, afin de rendre ces réalisations économiquement possibles mais aussi fiables, de concevoir une découpe fonctionnelle et une architecture de programmes applicables au problème des interfaces en général.

Ces problèmes ont été concrétisés par la présence de deux personnages : le parleur et l'auditeur entre qui l'adaptateur essaie de résoudre un problème d' incommunicabilité.

Les concepts dont il est question dans ce mémoire ont d'ores et déjà été appliqués à plusieurs dizaines de cas particuliers, certains ayant atteint une certaine maturité, d'autres restant en évolution.

Nous n'avons envisagé que le cas communication uni-directionelle en effleurant seulement les (difficiles !) problèmes des interfaces dans les cas de flux d'information bi-directionnels qui sont très fréquents dans la pratique.

Enfin, les perspectives ne manquent pas et concernent tout particulièrement une formalisation plus poussée des flux d'information et des protocoles qui tendraient vers une automatisation plus avancée de l'élaboration de logiciels.

BIBLIOGRAPHIE

<1> : Crocus,
Systèmes d'exploitation des ordinateurs (ouvrage collectif),
Dunod Informatique, 1976

<2> : C. Macchi, J-F Guilbert,
Téléinformatique,
Dunod Informatique, 1979

<3> : E.W. Dijkstra,
"The Structure of the "THE"-multiprogramming System",
CACM Volume 11-5, 1968

<4> : GA.D. Hall,
"The M6 Macroprocessor", Computing Science Tech. Rep. ≠ 2,
Bell Telephone Laboratories, 1969

<5> : MCS-85 User's Manual,
Intel Corporation, 1978

<6> : 8080/8085 Assembly Language Programming,
Intel Corporation, 1978

<7> : J.P. Tremblay, P.G. Sorenson,
An Introduction to Data Structures,
International Student Edition (Mc Graw-Hill), 1976